Exploring the Titanic Dataset

In [1]:

```
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import pandas_profiling
```

In [2]:

```
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
```

In [3]:

```
import warnings
warnings.filterwarnings('ignore')
```

We Import our Data

In [4]:

```
titanic = sns.load_dataset('titanic')
titanic.head()
```

Out[4]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

In [5]:

```
titanic.shape
```

Out[5]:

```
(891, 15)
```

In [6]:

```
titanic.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
survived	891.0	0.383838	0.486592	0.00	0.0000	0.0000	1.0	1.0000
pclass	891.0	2.308642	0.836071	1.00	2.0000	3.0000	3.0	3.0000
age	714.0	29.699118	14.526497	0.42	20.1250	28.0000	38.0	80.0000
sibsp	891.0	0.523008	1.102743	0.00	0.0000	0.0000	1.0	8.0000
parch	891.0	0.381594	0.806057	0.00	0.0000	0.0000	0.0	6.0000
fare	891.0	32.204208	49.693429	0.00	7.9104	14.4542	31.0	512.3292

In [7]:

```
titanic.dtypes
```

Out[7]:

```
survived          int64
pclass            int64
sex              object
age             float64
sibsp             int64
parch             int64
fare            float64
embarked         object
class          category
who              object
adult_male         bool
deck           category
embark_town      object
alive            object
alone              bool
dtype: object
```

In [8]:

```
# We determine the missing values and percentage missing values per feature

total = titanic.isnull().sum().sort_values(ascending=False)
percent = ((titanic.isnull().sum().sort_values(ascending=False))/len(titanic))*100
percent = round(percent, 2)
missing = pd.concat([total, percent], axis=1, keys=['missing val', 'percent'])
missing[missing['missing val']>0]
```

Out[8]:

	missing val	percent
deck	688	77.22
age	177	19.87
embark_town	2	0.22
embarked	2	0.22

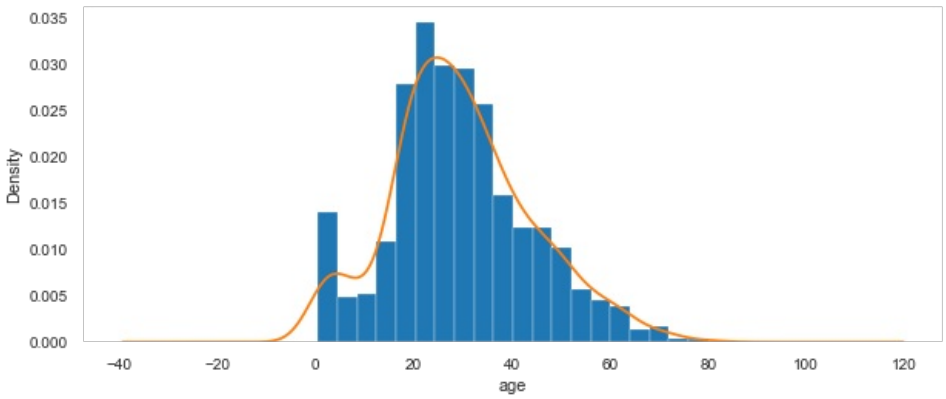
Analysis of the Feature Age

In [9]:

```
ax = titanic.age.hist(bins=20, figsize=(10, 4), density=True)
titanic.age.plot(kind='density')
ax.set_xlabel('age')
```

Out[9]:

```
Text(0.5, 0, 'age')
```

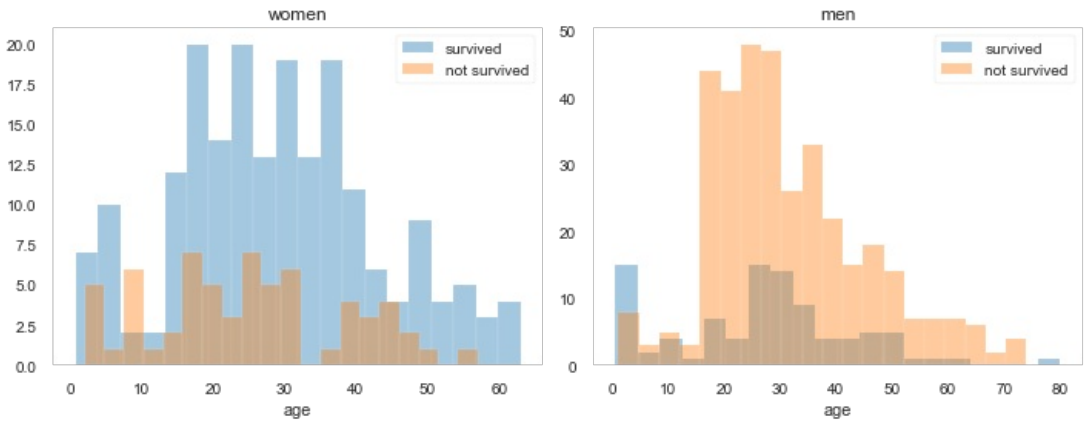


Here, we see that many of those who survived are children below 5 years and adults between ages 15 and 35.

Let us see the survival distribution of men, women and children.

In [10]:

```
survived = 'survived'
not_survived = 'not survived'

fig, axes = plt.subplots(1, 2, figsize=(10, 4))

women = titanic[titanic['sex']=='female']
men = titanic[titanic['sex']=='male']

ax = sns.distplot(women[women[survived]==1]['age'].dropna(), label=survived, bins=20, ax=axes[0], kde=False)
ax = sns.distplot(women[women[survived]==0]['age'].dropna(), label=not_survived, bins=20, ax=axes[0], kde=False)
ax.legend(loc='best')
ax.set_title('women')

ax = sns.distplot(men[men[survived]==1]['age'].dropna(), label=survived, bins=20, ax=axes[1], kde=False)
ax = sns.distplot(men[men[survived]==0]['age'].dropna(), label=not_survived, bins=20, ax=axes[1], kde=False)
ax.legend(loc='best')
ax.set_title('men')

fig.tight_layout()
```



We see that among women, those who survived were more than those who did not survive. Also, among female children less than 10, those who survived were more than those who did not survive.

We see that among men, those who did not survive were more than those who survived. However, among male children less than 5, those who survived were more than those who did not survive.

We see that most women who survived were between 15 and 40. Most women who died were between 18 and 35. Likewise, most men who survived were between 25 and 35 and most men who died were between 18 and 50.

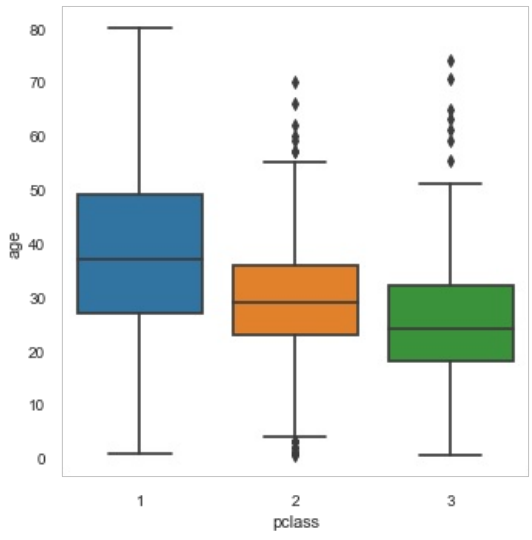
This shows that the age feature plays a part in determining who survived and who lived.

In [11]:

```
sns.catplot(x='pclass', y='age', data=titanic, kind='box')
```

Out[11]:

```
<seaborn.axisgrid.FacetGrid at 0x16f47ed0>
```



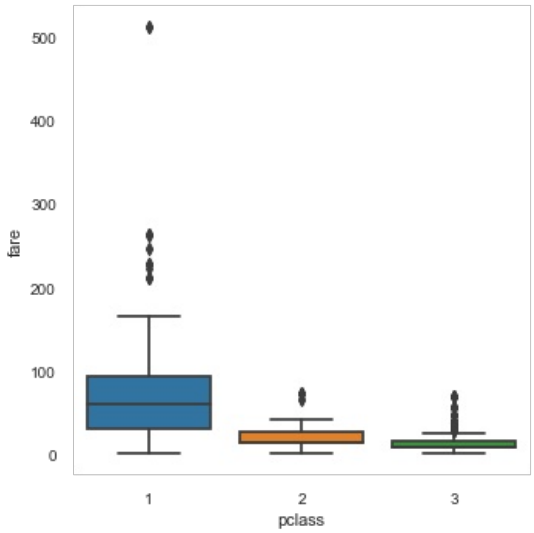
The diagram above shows that we have very young people in 3rd class and older people in 1st class. So age determined where a passenger would be.

In [12]:

```
sns.catplot(x='pclass', y='fare', data=titanic, kind='box')
```

Out[12]:

```
<seaborn.axisgrid.FacetGrid at 0x16f57e10>
```



The diagram shows that those in 1st class paid the most and those in 3rd class paid the least.

In [13]:

```
titanic.pivot_table('survived', index='pclass')
```

Out[13]:

pclass	survived
1	0.629630
2	0.472826
3	0.242363

The table shows that survival decreases from 1st class to 3rd class.

We can conclude from the diagrams that age determined the income of the passengers. The passenger incomes determined the class they boarded. The class they boarded determined whether they survived or not. So age is an essential feature.

We will consider the age feature and fill in the missing values.

In [14]:

```
titanic[titanic['pclass']==1]['age'].median()
```

Out[14]:

```
37.0
```

In [15]:

```
def fill_age(cols):
    age = cols[0]
    pclass = cols[1]
    if pd.isnull(age):
        if pclass==1:
            return titanic[titanic['pclass']==1]['age'].median()
        elif pclass==2:
            return titanic[titanic['pclass']==2]['age'].median()
        elif pclass==3:
            return titanic[titanic['pclass']==3]['age'].median()
    else:
        return age
```

In [16]:

```
# We fill in the missing values for age

titanic.age = titanic[['age', 'pclass']].apply(fill_age, axis=1)
```

In [17]:

```
# We have no missing values for age

titanic.age.isnull().sum()
```

Out[17]:

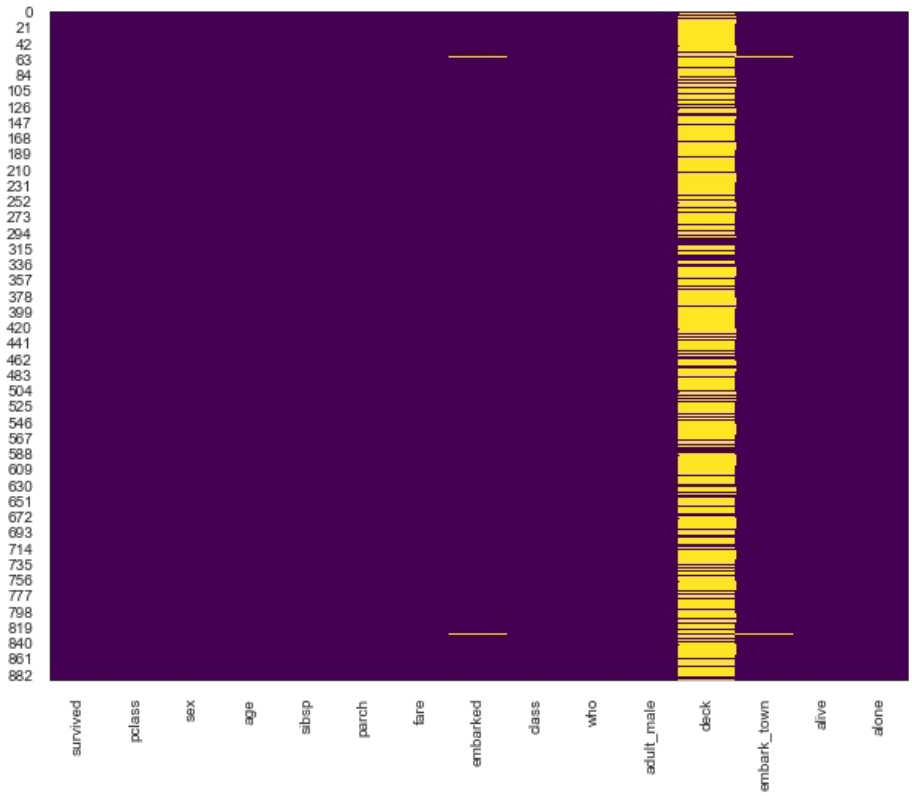
```
0
```

In [18]:

```
# The diagram below shows we have no missing values for age

plt.figure(figsize=(10, 8))
sns.heatmap(titanic.isnull(), cbar=False, cmap='viridis')
```

Out[18]:

```
<AxesSubplot:>
```



We have some missing values for embarked. Let us analyse this feature.

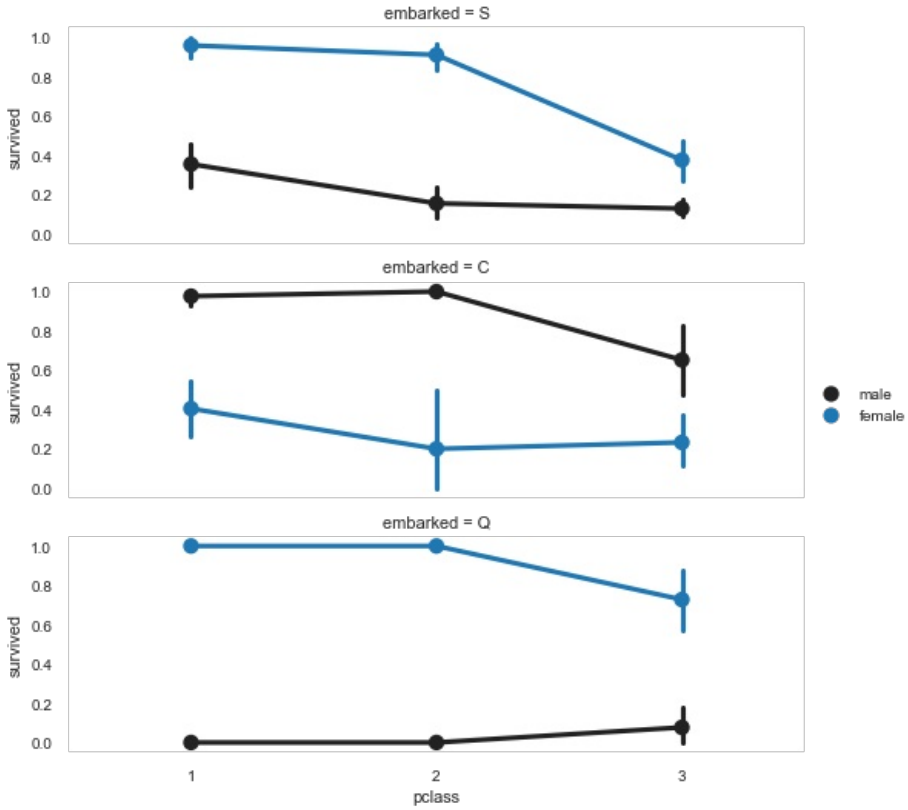
Analysis of the Feature Embarked

In [19]:

```
f = sns.FacetGrid(titanic, row='embarked', height=2.5, aspect=3)
f.map(sns.pointplot, 'pclass', 'survived', 'sex', order=None, hue_order=None)
f.add_legend()
```

Out[19]:

```
<seaborn.axisgrid.FacetGrid at 0x1c3332d0>
```



More women who embarked at ports S and Q survived than men who embarked at the same ports. More men who embarked at port C survived than women who embarked at the sampe port.

In [20]:

```
titanic.embarked.value_counts()
```

Out[20]:

```
S    644
C    168
Q     77
Name: embarked, dtype: int64
```

The highest number of passengers came from port S and the least came from port Q. We will fill the missing values with port S.

In [21]:

```
titanic.embarked.fillna('S', inplace=True)
titanic.embarked.isnull().sum()
```

Out[21]:

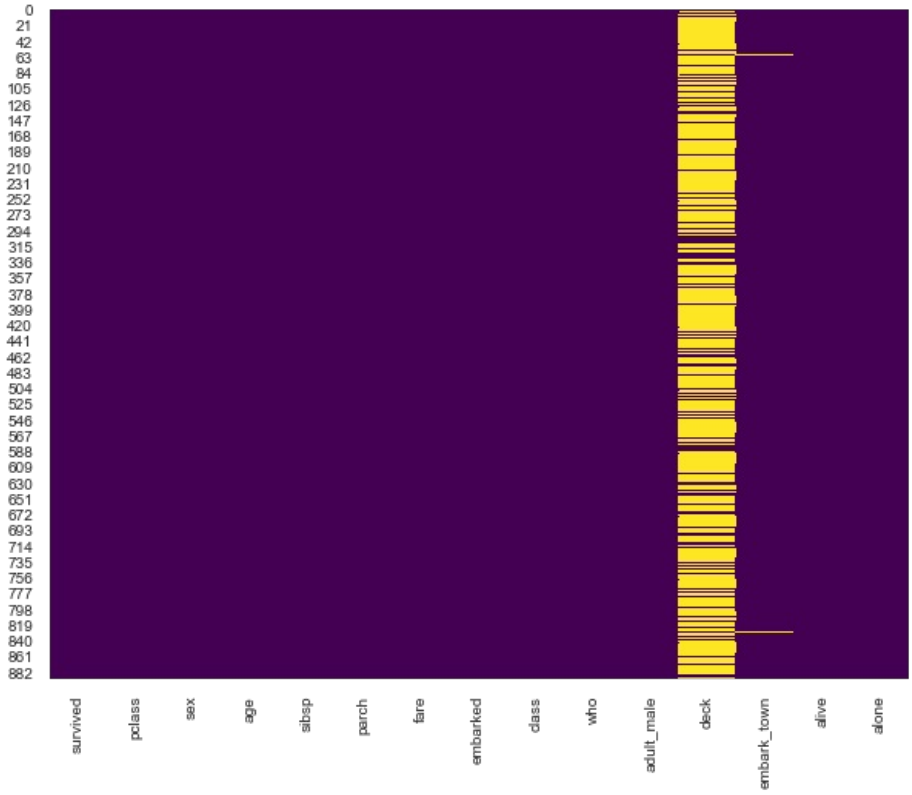
```
0
```

In [22]:

```
plt.figure(figsize=(10, 8))
sns.heatmap(titanic.isnull(), cbar=False, cmap='viridis')
```

Out[22]:

<AxesSubplot:>



From the diagram above, we do not have missing values in embarked anymore.

Analysing other Features

** More than 50% of the values of feature deck are missing. So we will drop this feature.*

** Embarked Town is a duplicate of embarked. So we will drop this feature.*

** Alive is a duplicate of survived. So we will drop this feature.*

** Adult male does not give an additional information.So we will drop this feature.*

** Class is a duplicate of pclass. So we will drop this feature.*

In [23]:

```
titanic.drop(['class', 'adult_male', 'deck', 'embark_town', 'alive'], axis=1, inplace=True)
```

In [24]:

```
titanic.head()
```

Out[24]:

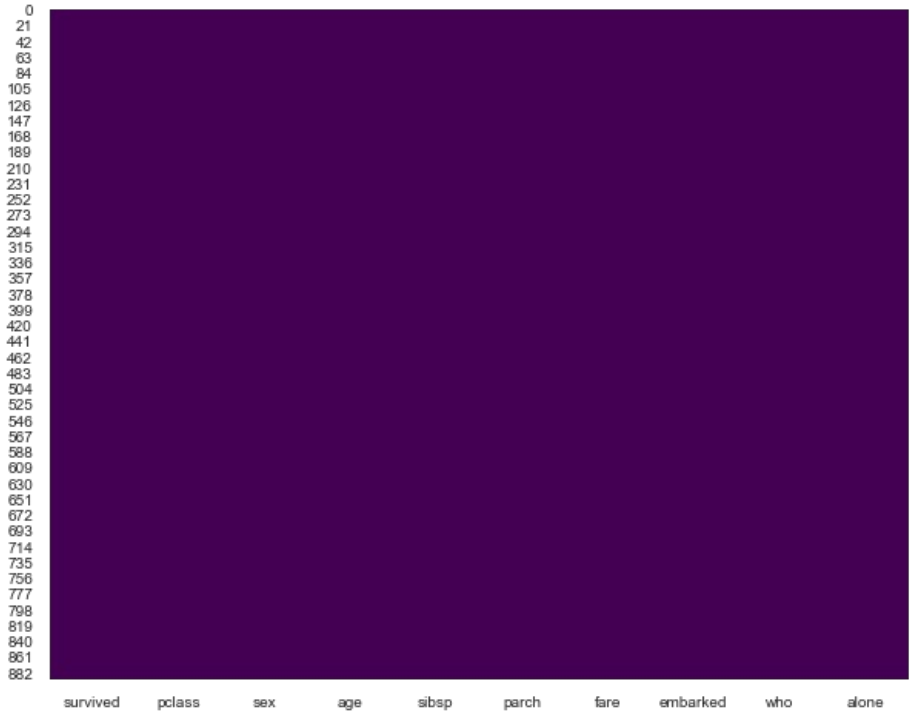
	survived	pclass	sex	age	sibsp	parch	fare	embarked	who	alone
0	0	3	male	22.0	1	0	7.2500	S	man	False
1	1	1	female	38.0	1	0	71.2833	C	woman	False
2	1	3	female	26.0	0	0	7.9250	S	woman	True
3	1	1	female	35.0	1	0	53.1000	S	woman	False
4	0	3	male	35.0	0	0	8.0500	S	man	True

In [25]:

```
plt.figure(figsize=(10, 8))
sns.heatmap(titanic.isnull(), cbar=False, cmap='viridis')
```

Out[25]:

```
<AxesSubplot:>
```



The diagram above shows we do not have any missing values in our data.

In [26]:

```
titanic.dtypes
```

Out[26]:

```
survived      int64
pclass        int64
sex          object
age         float64
sibsp         int64
parch         int64
fare        float64
embarked     object
who          object
alone          bool
dtype: object
```

In [27]:

```
titanic['age'] = titanic['age'].astype('int')
titanic['fare'] = titanic['fare'].astype('int')
```

In [28]:

```
titanic.head()
```

Out[28]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	who	alone
0	0	3	male	22	1	0	7	S	man	False
1	1	1	female	38	1	0	71	C	woman	False
2	1	3	female	26	0	0	7	S	woman	True
3	1	1	female	35	1	0	53	S	woman	False
4	0	3	male	35	0	0	8	S	man	True

Encoding our Data

In [29]:

```
titanic_enc = pd.get_dummies(titanic, columns=['pclass', 'sex', 'embarked', 'who', 'alone'])
titanic_enc.head()
```

Out[29]:

	survived	age	sibsp	parch	fare	pclass_1	pclass_2	pclass_3	sex_female	sex_male	embarked_C	embarked_Q	embarked_S	who_
0	0	22	1	0	7	0	0	1	0	1	0	0	1	
1	1	38	1	0	71	1	0	0	1	0	1	0	0	
2	1	26	0	0	7	0	0	1	1	0	0	0	1	
3	1	35	1	0	53	1	0	0	1	0	0	0	1	
4	0	35	0	0	8	0	0	1	0	1	0	0	1	

Splitting our Data and Training our Model

In [30]:

```
X = titanic_enc.drop(['survived'], axis=1)
y = titanic.survived
```

In [31]:

```
X.shape, y.shape
```

Out[31]:

```
((891, 17), (891,))
```

In [32]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=0, stratify=y)
```

In [33]:

```
X_train.shape, X_test.shape
```

Out[33]:

```
((596, 17), (295, 17))
```

In [35]:

```
# Random Forest Classifier

scores = cross_val_score(RandomForestClassifier(), X_train, y_train, cv=10, n_jobs=-1)
score_rfc = scores.mean()
score_rfc
```

Out[35]:

```
0.7953107344632768
```

In [36]:

```
# Gradient Boosting Classifier

scores = cross_val_score(GradientBoostingClassifier(), X_train, y_train, cv=10, n_jobs=-1)
score_gbc = scores.mean()
score_gbc
```

Out[36]:

```
0.8154519774011298
```

In [37]:

```
# Logistic Regression

scores = cross_val_score(LogisticRegression(), X_train, y_train, cv=10, n_jobs=-1)
score_logreg = scores.mean()
score_logreg
```

Out[37]:

```
0.8087005649717515
```

In [38]:

```
# Stochastic Gradient Classifier

scores = cross_val_score(SGDClassifier(), X_train, y_train, cv=10, n_jobs=-1)
score_sgd = scores.mean()
score_sgd
```

Out[38]:

```
0.6707909604519774
```

In [39]:

```
# Linear SVC

scores = cross_val_score(SVC(), X_train, y_train, cv=10, n_jobs=-1)
score_svc = scores.mean()
score_svc
```

Out[39]:

```
0.6878248587570621
```

In [40]:

```
# Decision Tree Classifier

scores = cross_val_score(DecisionTreeClassifier(), X_train, y_train, cv=10, n_jobs=-1)
score_tree = scores.mean()
score_tree
```

Out[40]:

```
0.7886440677966101
```

In [41]:

```
# Naive Bayes

scores = cross_val_score(GaussianNB(), X_train, y_train, cv=10, n_jobs=-1)
score_gnb = scores.mean()
score_gnb
```

Out[41]:

```
0.7968361581920903
```

In [42]:

```
# KNN

scores = cross_val_score(KNeighborsClassifier(), X_train, y_train, cv=10, n_jobs=-1)
score_knn = scores.mean()
score_knn
```

Out[42]:

```
0.7298587570621468
```

In [44]:

```
scores = pd.DataFrame(
    {'Models':['Random Forest Classifier', 'Gradient Boosting Classifier', 'Logistic Regression', 'SGDClassifier'
, 'Linear SVC', 'Decision Tree Classifier', 'Naive Bayes', 'KNN'],
     'score':[score_rfc, score_gbc, score_logreg, score_sgd, score_svc, score_tree, score_gnb, score_knn]})
scores.sort_values(by='score', ascending=False)
```

Out[44]:

	Models	score
1	Gradient Boosting Classifier	0.815452
2	Logistic Regression	0.808701
6	Naive Bayes	0.796836
0	Random Forest Classifier	0.795311
5	Decision Tree Classifier	0.788644
7	KNN	0.729859
4	Linear SVC	0.687825
3	SGDClassifier	0.670791

In [47]:

```
gbc = GradientBoostingClassifier().fit(X_train, y_train)
gen_score = gbc.score(X_test, y_test)
print('Generalization score:', gen_score)
```

```
Generalization score: 0.847457627118644
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: